Cheat sheet: 10 AngularJS security best practices

snyk

1. The "Angular way" safeguards you from XSS

AngularJS by default applies automatic output encoding and input sanitization that is context-aware for any data values that are pushed on to the DOM.

Interpolate data the "Angular way" with **ngBind** directives or curly braces such as **{{ data }}** so that Angular applies its output encoding and sanitization by default.

2. Avoid using the Angular DOM-related input injection

Avoid using the Angular DOM-related input injection which may introduce vulnerabilities:

- in Angular 1.2 and prior avoid using the **ng-bind-html-unsafe** directive.
- in Angular 1.2 and later avoid blindly trusting user input with Strict Contextual Escaping collection of methods such as **$sce.trustAsHtml(value).**

3. Avoid dynamically loading Angular templates from untrusted sources

It is best to avoid dynamically loading Angular templates from untrusted sources. Follow these practices when in need to setup resource whitelists for **$sceDelegateProvider.resourceUrlWhitelist()**:

- use HTTPS as a secure medium to fetch the remote templates and ensure up to date TLS configuration exists on the remote endpoint.
- avoid using the double asterisk wildcard ** for domains or protocols
- when necessary, create a blacklist for defense-in-depth.

4. AngularJS open redirect vulnerabilities

Avoid open direct pitfalls by implementing user-provided input directly to perform page navigation.

- Avoid patterns such as window.location.href = $location.hash which potentially lead to JavaScript Code Injection attacks.
- Use dictionary maps to perform page navigation based on user-provided input.

5. Server-side Angular code injection

Mitigate server-side Angular code injection:

- Avoid mixing server-side and client-side templates. Instead, treat templates only within one application context: either the server-side or the client-side.
- Reduce the scope of **ng-app** directive from an HTML's document body to specific DOM element context within the page itself.
- Bind the data from the template to **ng-bind** or **ng-bind-html** to ensure user input is being properly handled with Angular's support for output encoding and sanitization controls with these Angular directives.
- Use **ng-non-bindable** to make sure the data is not being treated by Angular as an expression that needs to be evaluated and so mitigating the Angular code injection.

6. Avoid using the Angular .element jQuery-compatible API to manipulate the DOM

Avoid using Angular's **angular.element()** jQuery-compatible API to manipulate the DOM as this leads to potential Cross-site Scripting vulnerabilities due to directly creating HTML elements on the DOM.

Angular provides an **angular.element()** API to provide jQuery-like API compatibility (jqLite) to query and manipulate the DOM directly.

This might seem as a convenient way to access and interact with the DOM, however, it introduces potential injection vulnerabilities due to the unsafe DOM injection.

7. Use Angular security linters

Use static code analysis tools to automate finding insecure code and alerting developers when this happens early in the development process. Security linters that are used for Angular secure coding practices:

- eslint-plugin-scanjs-rules
- eslint-plugin-angular

8. Scan and fix vulnerabilities in Angular third-party components

AngularJS has over 20 vulnerabilities to date and there are Angular components with millions of downloads that are still vulnerable.

1. Connect Snyk to GitHub or other SCMs for optimal CI/CD integration with your projects.
2. Snyk finds vulnerabilities in 3rd party components you use and opens fix Pull Requests so you can merge the version update and mitigate the risk.

9. Built-in CSRF support for Angular applications

Angular has built-in support for CSRF token handling on the client-side via its $http service. Use this service instead of rolling your own.

10. Angular's built-in CSP compatibility

Implementing a Content Security Policy (CSP) is an important step in providing an additional layer of security, especially one that helps mitigate Cross-site Scripting attacks.

Some Angular features conflict with common CSP restrictions such as those requiring no inline JavaScript code evaluation. However, instead of turning CSP off completely, which is not recommended, Angular has the ngCsp directive that offers compatibility with CSP configuration.

Author

@liran_tal
Node.js Security WG & Developer Advocate at Snyk